Use Case Analysis based on Formal Methods: An Empirical Study

Marcos Oliveira Junior, Leila Ribeiro, Érika Cota,
Lucio Mauro Duarte, Ingrid Nunes, and Filipe Reis *

PPGC - Institute of Informatics – Federal University of Rio Grande do Sul (UFRGS)
PO Box 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{marcos.oliveira,leila,erika,lmduarte,ingridnunes,freis}@inf.ufrgs.br

Abstract. *Use Cases (UC)* are a popular way of describing system behavior and UC quality impacts the overall system quality. However, they are presented in natural language, which is usually the cause of issues related to imprecision, ambiguity, and incompleteness. We present the results of an empirical study on the formalization of UCs as Graph Transformation models (GTs) with the goal of running tool-supported analyses on them and revealing possible errors. To evaluate our approach, we apply it to a set of real UC descriptions obtained from a software developer company and measured the results through metrics. The final results demonstrate that this approach can reveal real problems that could otherwise go undetected and, thus, help improve the quality of the UCs.
Keywords. Use Cases, Graph Transformation, Model Analysis.

1 Introduction

Use Cases (UC) [2] are a popular model for documenting software expected behaviour. In current practice, UC descriptions are typically informally documented using, in most cases, natural language in a predefined structure. Being informal descriptions, UCs might be ambiguous and imprecise. Thus, the verification of UCs normally corresponds to manual inspections and walkthroughs [4], and detecting problems is not a trivial task. Since software quality is highly dependent on the quality of the specification, cost-effective strategies to decrease the number of errors in UCs are crucial. Strategies for the formalization of UCs have already been proposed, however, many of them assume a particular syntax for UC description tailored for their particular formalisms. This limits the expression of requirements and, in some cases, also restrains the semantics of the UC. Our aim is to keep the expressiveness of a description in natural language and use a formalism for modeling/analysing UCs that is flexible enough to maintain the semantics defined by stakeholders.

In this paper, we investigate the suitability of Graph Transformation (GT) as a formal model to describe and analyze UCs. Some reasons for choosing GT are: the elements of a UC can be naturally represented as graphs; it is a visual

* This work is partially supported by the VeriTeS project (FAPERGS and CNPq).

language; the semantics is very simple yet expressive; GT is data-driven; there are various static and dynamic analysis techniques available for GT, as well as tools to support them (e.g., [10]). We work towards an approach that integrates UC formalization and tool-supported analysis, with the objective of improving the quality of UCs. We applied our approach on a set of real UC descriptions obtained from a software development company and measured the results.

This paper is organized as follows: Section 2 presents the necessary background information and a overview of the translation of UCs for GTs.; Section 3 presents the settings of the conducted empirical study; Section 4 presents an analysis and discussion of results; Section 5 presents an analysis of threats to the study; Section 6 presents a comparative analysis of our technique with some related work; and Section 7 concludes the paper and discusses future work.

2 Modeling UCs using GTs

2.1 Background

Use Cases. According to Cockburn (2000) [2], a *Use Case (UC)* defines a contract between stakeholders of a system, describing part of the system behavior. The main purpose of a UC description is the documentation of the expected system behavior so as to ease the communication between stakeholders, often including non-technical people, about required system functionalities. For this reason, UC descriptions are usually described in a textual form.
Graph Transformations. The formalism of *Graph Transformations (GT)* [7] is based on defining states of a system as graphs and state changes as rules that transform these graphs. Our analysis of GTs is based on concurrent rules and critical pairs, two methods of analysis independent from the initial state of the system and, thus, they are complementary to any other verification strategy based on initial states (such as testing).

2.2 Proposed Formalization and Verification Approach

The proposed approach, detailed in [6], takes as input a textual UC description, from which the entities and actions that will be part of the formal model are identified. Then, basic verifications can be performed regarding the consistency of the extracted information. If inconsistencies are detected, the UC must be rewritten to eliminate them or the analyst can annotate the problem to be resolved later on. When no basic inconsistencies are found, the GT can then be generated. In this process, conditions and effects of actions are modeled as states and a type graph is built. After that, each UC step is modeled as a transition rule from one state (graph) to another. Having the GT, a series of automatic verifications can be performed to detect possible problems.

We use the AGG tool [10] to perform the automatic analyses on the GT model. All detected issues are annotated as *open issues (OIs)* along with the solutions (when applicable). Open issues are classified according to their severity

level: Yellow (for warnings), Orange (for relevant issues), or Red (for critical issues). The actions to be taken regarding found OIs depend on the analysts, who can determine whether an OI is in fact a real problem.

3 Empirical Study Settings

In order to adequately evaluate our approach, we followed the principles of Experimental Software Engineering [11] and the GQM template [1]. Our main study goal was to demonstrate the usefulness of GTs to improve the quality of UCs by the identification of OIs, from a perspective of the researcher, in the context of a single real software development project. From this, we derived two research questions, which we aimed to answer with our study.

RQ-1 Are system analysts able to detect problems in their own UC descriptions without additional support?

RQ-2 How effective is our GT-based approach in identifying problems in UCs?

The UC descriptions we used in our study are part of the analysis documentation of an industrial software project. This project involves the development of a typical system to manage products from a warehouse, with functional requirements such as adding new products, creating sale orders, and releasing products in stock. We do not provide any further details about our target system and its UCs due to a confidentiality agreement.

3.1 Procedure

The procedure of the study consists of the following steps:

(1) Analysis by System Analyst. We requested a system analyst responsible for the UC descriptions to carefully revise them, and point out problems, such as ambiguity, imprecision, omission, incompleteness, and inconsistency.

(2) UC Formalization. Given a set of UCs, we performed the steps detailed in [6] to formalize them using GTs and used the AGG tool to analyze them, detecting some OIs.

(3) Evaluation of Open Issues. After identifying OIs, we had evaluated whether detected OIs were real problems in the analyzed UCs.

(4) Data Analysis. Our aim is that our approach detects all and only real problems. This can be seen as a *classification problem*, and thus the effectiveness of our approach can be measured using the metrics, widely used in the context of information retrieval, of *precision* and *relative recall* [5], whose formulas are shown below, where *true positives* are OIs that correspond to real problems; $false\ positives$ are OIs that are not real problems; and $false\ negatives$ are real problems not identified as OIs.

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{1}$$

$$RelativeRecall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2}$$

Table 1: Study results

OI Type	UC 1		UC 2		UC 3		UC 4		UC 5		Total	
	#OI	#P	#OI	#P	#OI	#P	#OI	#P	#OI	#P	#OI	#P
⚠	3	2	4	2	2	1	4	2	1	0	14	7
?	1	1	1	1	1	1	0	0	2	2	5	5
✋	3	3	1	1	3	2	3	3	3	3	13	12
Total	**7**	**6**	**6**	**4**	**6**	**4**	**7**	**5**	**6**	**5**	**32**	**24**
Legend: UC - Use Case; OI - Open Issue; P - Problem.												

4 Results and Discussion

After revising the original UCs, the system analyst found no problems. However, after applying our approach to these UCs, we identified 32 OIs across the 5 UCs, which gives an average of 6.4 OIs per UC. This is an expressive number, given that the system analyst stated that the UCs had been correctly specified. In order to verify whether the identified OIs were false alarms (false positives), the system analyst was asked to check each one of them. Of the 32 OIs, 24 were pointed out as real problems and only 8 as false positives.

Table 1 presents our results in detail. It shows the number of OIs found in each UC (columns labeled with OI) and how many of these OIs were confirmed as real problems (columns labeled with P). The rows show the number of detected OIs with respect to their level of severity. The table also presents the total number of detected OIs and the total number of real problems considering all the 5 UCs. The symbols ⚠, ?, and ✋ represent warnings (severity Yellow), relevant issues (severity Orange), and critical issues (severity Red), respectively.

We then analyzed these results according to the selected metrics. Because the system analyst was unable to identify any problem without support, the number of problems not identified by our approach was 0, leading to *relative recall* = 1.0. As for the Precision, we obtained 0.75 (24 true positives and 8 false positives) — that is, 75% of the OIs identified by our GT-based approach were real problems. Not only most of the identified issues were actual problems, but also most of the false alarms (7 of 8) were related to low severity OIs.

By analyzing OIs not identified as problems, we observed that 6 of them were not necessarily classified as a false positive by the system analyst. They preferred to leave such issues as they were and postpone changes to future design decisions, considering that they alone could not decide what was the best approach to tackle those issues. The other 2 OIs found, confirmed as false positives, were related to incompleteness or ambiguities due to the lack of knowledge of the modeler about the problem domain and the internal processes of the company.

Note that OIs were identified without the intervention of any stakeholder. The only provided input was the software documentation in the form of UC descriptions and the output was a checklist with OIs to be revised. More importantly, had these problems been detected before the design and implementation, when they should have, development costs could have been potentially reduced.

5 Threats to Validity

During our study we carefully considered validity concerns. This section discusses the main threats we identified to validate this study and how we mitigated them.

Internal Validity. The main threat to internal validity of this study was the selection of modeler of UCs in the formalism of graphs. However, we want to show that, correctly following the steps of our strategy, the modeler does not need a deep understanding of the formalism. Moreover, we used the AGG tool to automate the analyses and provide a graphical interface.

Construct Validity. There are different ways of modeling a system through the formalism of graphs that can produce some threats to construct validity. The modeler may not follow correctly the modeling steps, being influenced by their prior knowledge about the formalism. Therefore, we proposed a roadmap, step by step, on how to model UCs as GTs, for both beginners and experts users.

Conclusion Validity. As the main threat to validity of the conclusion we highlight potential problems in the generation of the model in the formalism of graphs. Once again, our step-by-step modeling process should be followed to prevent the modeler from creating a model that is not consistent with the textual description. Moreover, the tool-supported verifications can also detect such modeling errors, thus reducing the risk of this threat.

External Validity. The main threat to the external validity was the selection of artifacts on which we based our study. We did not use any criteria to select either the project or the system analyst who participated of our study. We were aware of this threat during the study. However, we opted for randomly choosing artifacts to support the applicability of our strategy in different scenarios.

6 Related Work

Some authors have developed approaches for translating UCs to well-known formalisms, such as LTS [8], Petri Nets [12], and FSM [9]. Unlike these formalisms, a GT model is data-driven and we do not need to explicitly determine the control flow unless it is necessary to guarantee data consistency. The approach presented in [13] allows the simulation of the execution of the system but do not report the use of any type of analysis, which, in our opinion, reduces the advantage of having a formal model. The work described in [3] considers analyses such as critical pairs and dependencies involving multiple UCs and provides some ideas on the interpretation of the results. However, we propose a more structured way of providing diagnostic feedback about single UCs, which serves as a guide to point out the possible errors as well as their severity level.

7 Conclusions and Future Work

We investigated the suitability of GT as a formal basis for UC description and improvement. We evaluated our approach through an experiment with real software artifacts, where we could detect existing errors, which helped improve the

original UCs. Making a general analysis of the experiment, we consider the results promising, since it was possible to identify a large number of real problems based on a documentation that was produced at an early stage of software development. We observed the need for further automating the process, if not all, at least some steps, which is one of the most immediate planned future work.

A inter-UC analysis is currently being implemented as well as a more detailed diagnostic feedback. Within the same model frame, other types of validation and verification techniques on GT models, such as test case generation, model checking, and theorem proving, are also subject of current work. We plan to investigate whether we could reduce the impact and cost of changes by identifying which parts of the description are affected. Finally, note that, although we did not present any new formal method or verification technique here, a considerable amount of expertise in formal methods was required to define the OIs: they are meant to bridge the gap between the informal and formal worlds. We believe that this type of work is crucial towards the industrial adoption of formal methods.

References

1. Basili, V., Caldiera, C., Rombach, H.: Goal Question Metric Paradigm, Encyclopedia of Software Engineering, vol. 1. John Wiley & Sons (1994)
2. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (2000)
3. Hausmann, J.H., Heckel, R., Taentzer, G.: Detection of conflicting functional requirements in a use case-driven approach: A static analysis technique based on graph transformation. In: Proc. of the 24th ICSE. pp. 105–115 (2002)
4. Myers, G., Sandler, C., Badgett, T.: The Art of Software Testing. ITPro Collection, Wiley (2011)
5. Powers, D.M.: Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. Tech. Rep. SIE-07-001, Flinders University of South Australia (2007)
6. Ribeiro, L., Cota, E., Duarte, L.M., Oliveira Jr., M.A.d.: Improving the quality of use cases via model construction and analysis. In: Proc. of the 22nd WADT (2014)
7. Rozenberg, G. (ed.): Handbook of graph grammars and computing by graph transformation: volume I: Foundations. World Scientific, River Edge, USA (1997)
8. Sinnig, D., Chalin, P., Khendek, F.: LTS semantics for use case models. In: Proc. of the ACM SAC. pp. 365–370. ACM (2009)
9. Sinnig, D., Chalin, P., Khendek, F.: Use case and task models: An integrated development methodology and its formal foundation. ACM ToSEM 22(3), 27:1–27:31 (Jul 2013)
10. Taentzer, G.: AGG: A tool environment for algebraic graph transformation. In: Applications of Graph Transformations with Industrial Relevance, LNCS, vol. 1779, pp. 481–488. Springer Berlin Heidelberg (2000)
11. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Berlin Heidelberg (2012)
12. Zhao, J., Duan, Z.: Verification of use case with petri nets in requirement analysis. In: Proc. of the ICCSA: Part II. pp. 29–42. Springer-Verlag (2009)
13. Ziemann, P., Hǎvlscher, K., Gogolla, M.: From UML models to graph transformation systems. ENTCS 127(4), 17 – 33 (2005)